Recursion

Bryce Boe

2013/11/18

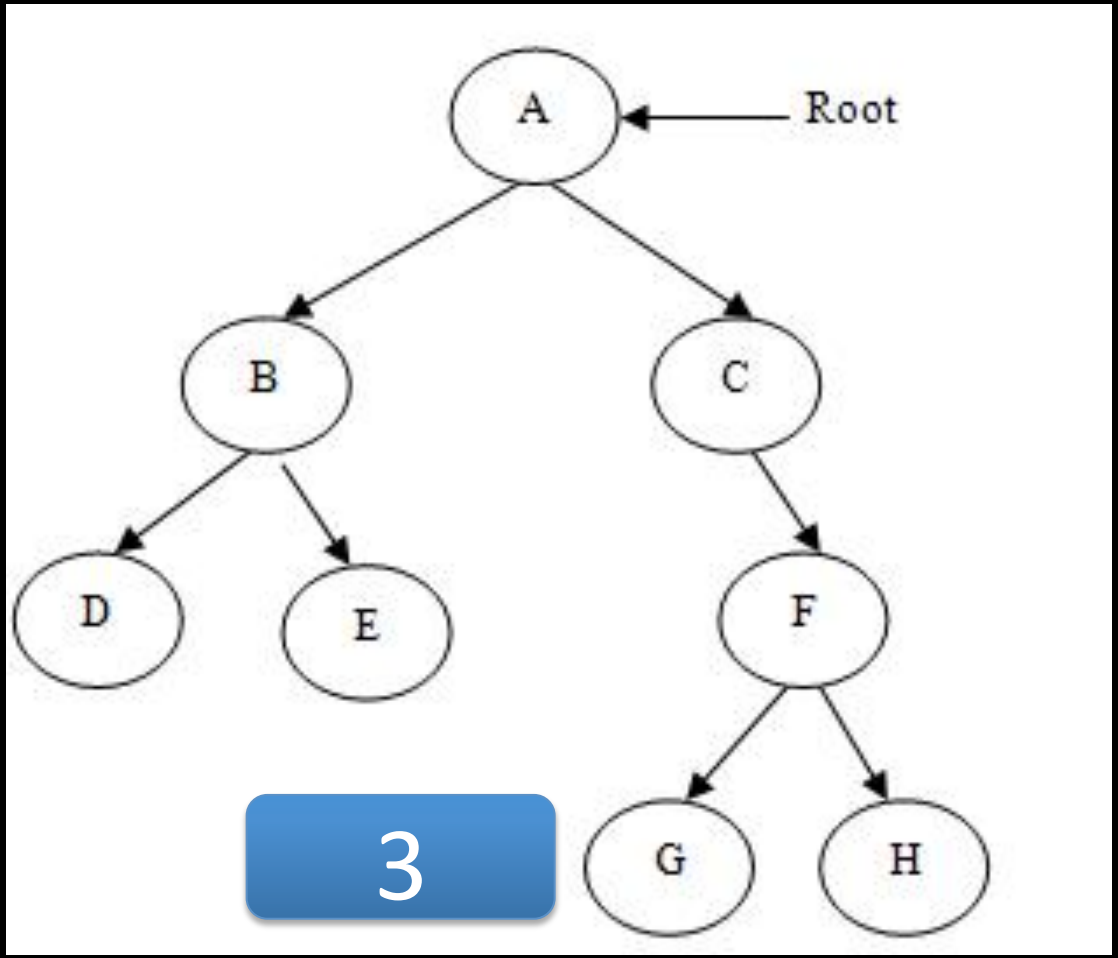
CS24, Fall 2013

Outline

- Wednesday Recap
- Lab 7 Iterative Solution
- Recursion
- Binary Tree Traversals
- Lab 7 Recursive Solution
- Merge Sort
- BST Remove

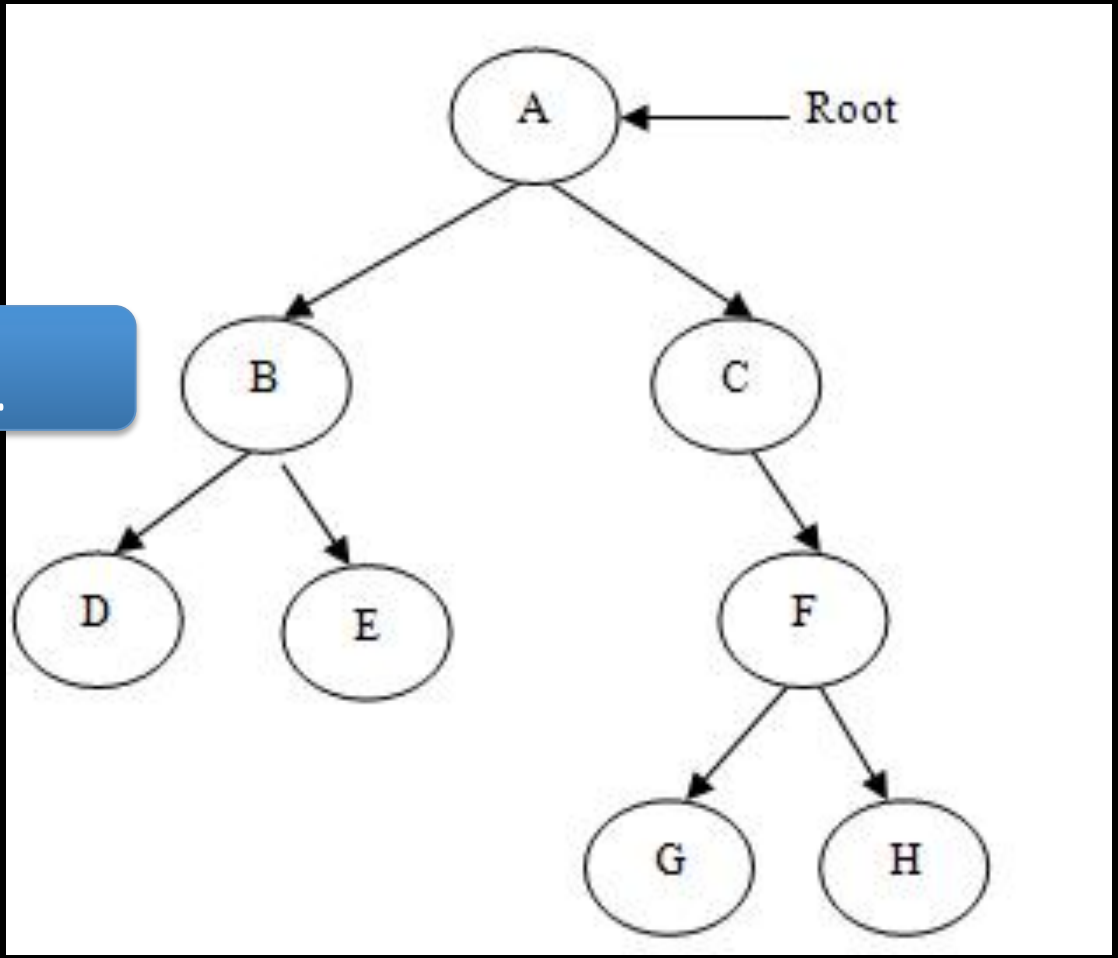
WEDNESDAY RECAP

What is the depth of G?

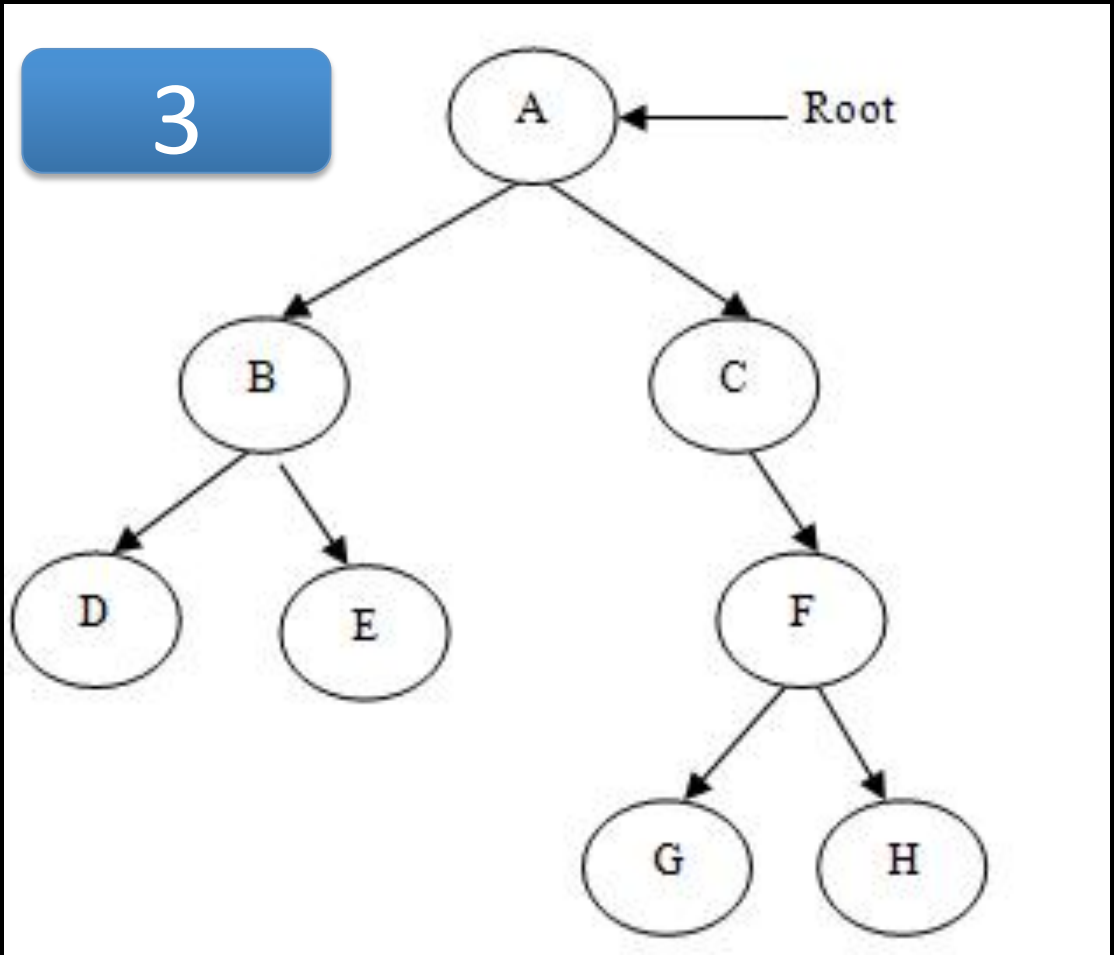


What is the height of B?

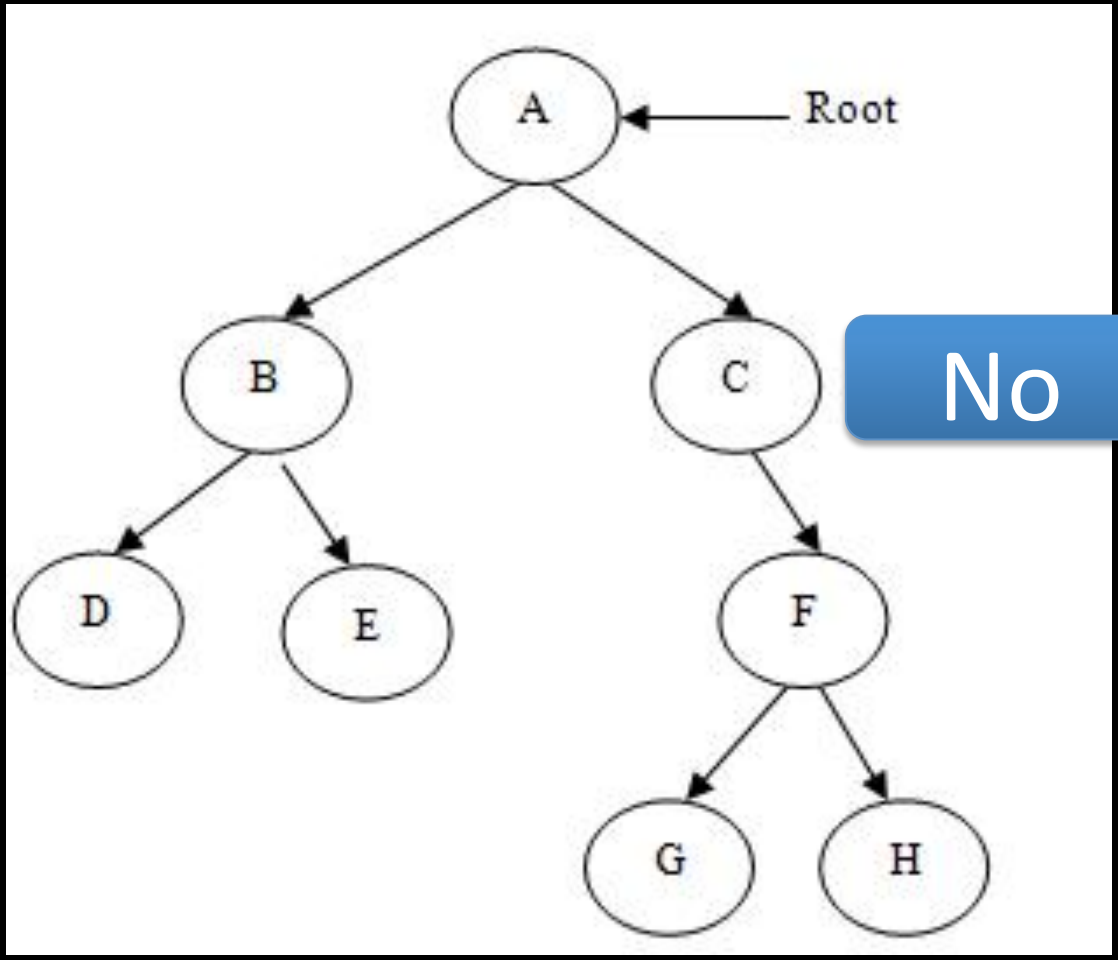
1



What is the height of the tree?



Is the tree balanced?



No

LAB 7 SOLUTION

RECURSION

What is recursion?

- The process of solving a problem by dividing it into similar subproblems
- Examples
  - Factorial: 5! = 5 * 4! = 5 * 4 * 3!
  - Length of linked list: L(node) = 1 + L(node->next)
  - Fibonacci Numbers: F(N) = F(n-1) + F(n-2)

Factorial

- Base Case:
  - F(1) = 1
- General Case
  - F(n) = n * F(n-1)

Factorial

```
int factorial(n) {
        if (n < 1) throw 1;  // Error condition
        else if (n == 1)  // Base Case
                return 1;
        else  // General Case
                return n * factorial(n – 1);
}
```

Linked List Length

- Base Case:
  - Length(last node) = 1
- General Case:
  - Length(node) = 1 + Length(node->next);

Linked List Length (option 1)

```
int length(Node *n) {
        if (n == NULL)  // Base Case
                return 0;
        else  // General Case
                return 1 + length(n->next);
}
```

Linked List Length (option 2)

```
int length(Node *n) {
        if (n == NULL) throw 1; // Error condition
        else if (n->next == NULL)  // Base Case
                return 1;
        else  // General Case
                return 1 + length(n->next);
}
```

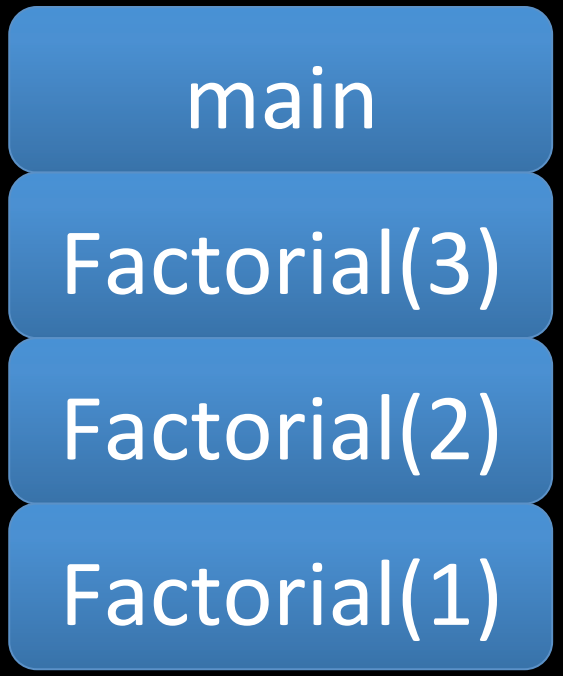
How much space is required for a list with 8 items?

Fibonacci Numbers

- Base Cases:
  - F(0) = 0
  - F(1) = 1
- General Case:
  - F(n) = F(n-1) + F(n-2)

Recursion and the Stack Segment

- main calls Factorial(3)



Recursion question

- How many activation records are created when calling Fibonacci(0)? 1
- Fibonacci(1)? 1
- Fibonacci(2)? 3
- Fibonacci(3)? 5
- Fibonacci(4)? 9
- Fibonacci(5)? 15

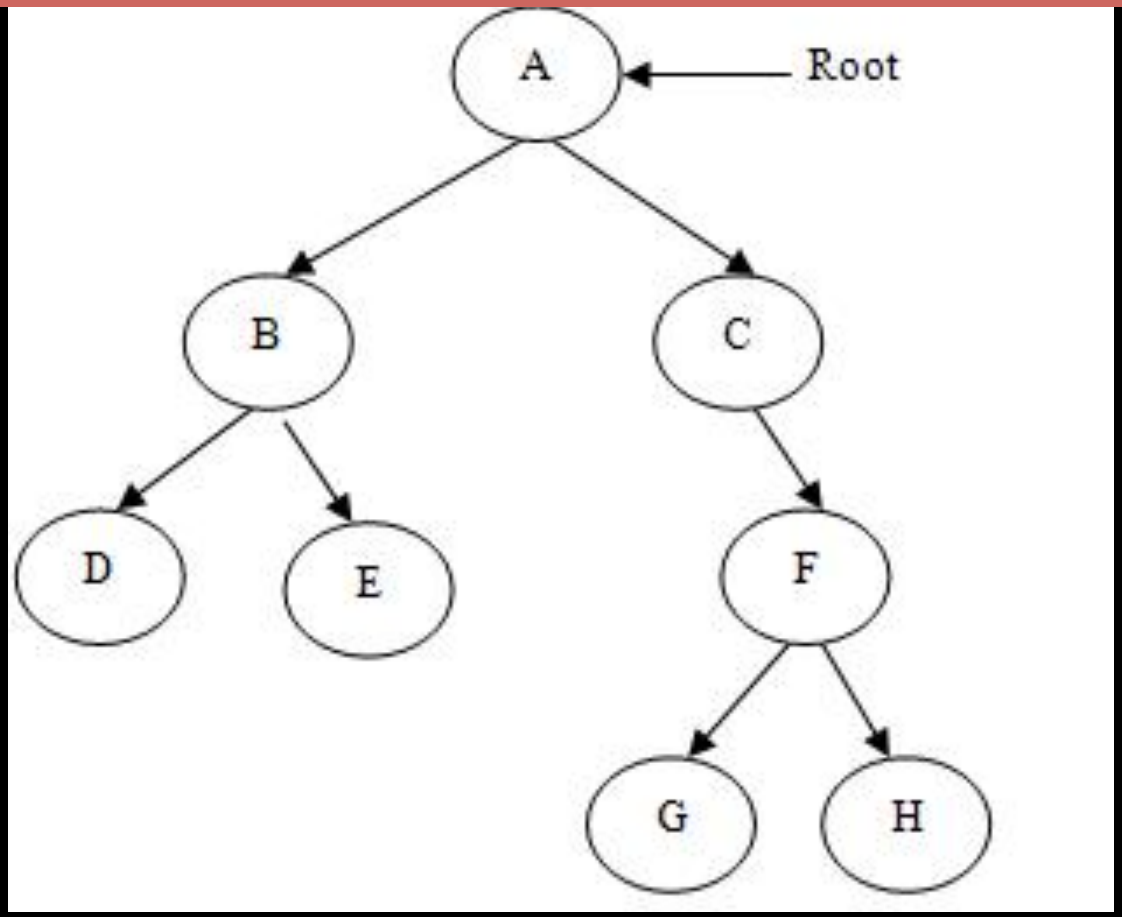
BINARY TREE TRAVERSALS

Depth-First Tree Traversals

- Can be done iteratively (with a stack) or recursively
- Pre-order
  - *Process* the node, recurse on left subtree, recurse on right subtree
- In-order
  - Recurse on the left subtree, *process* the node, recurse on the right subtree
- Post-order
  - Recurse on the left subtree, recurse on the right subtree, *process* the node

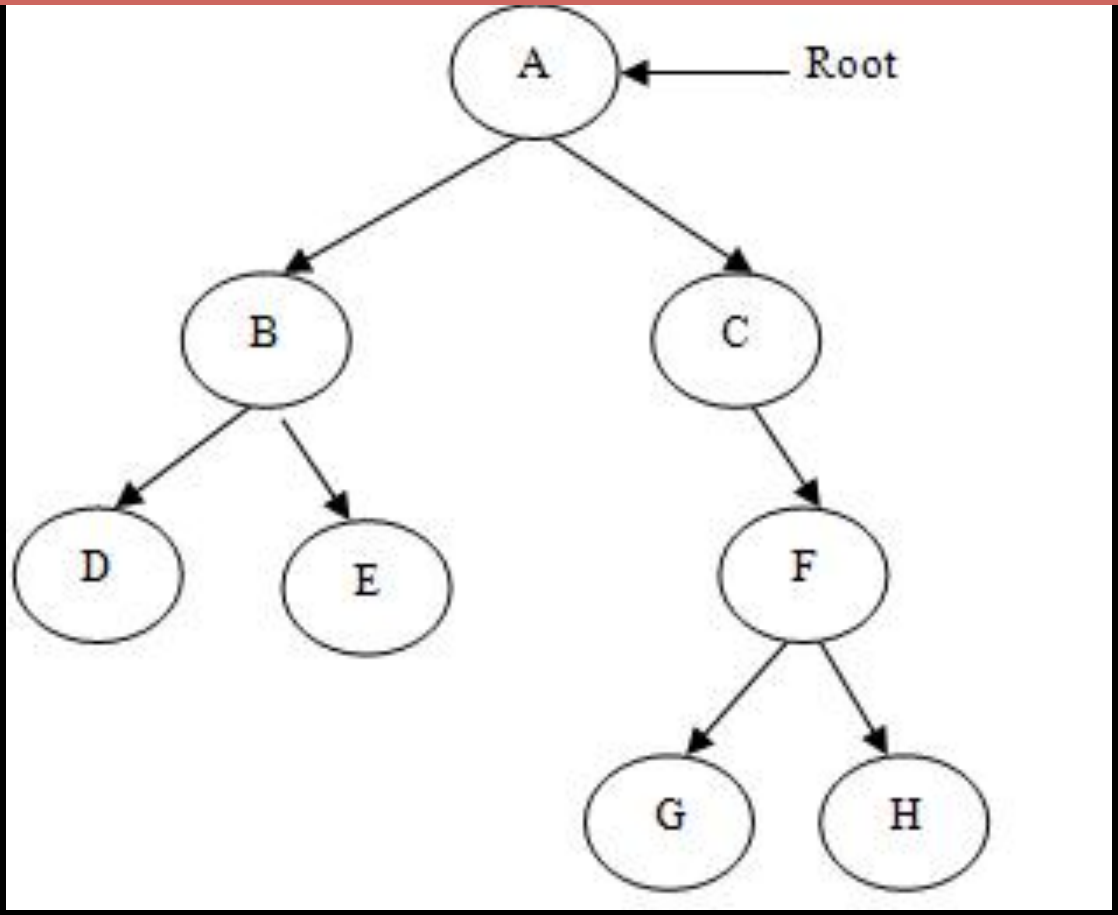
Pre-Order Traversal

A -> B - > D -> E -> C -> F -> G -> H



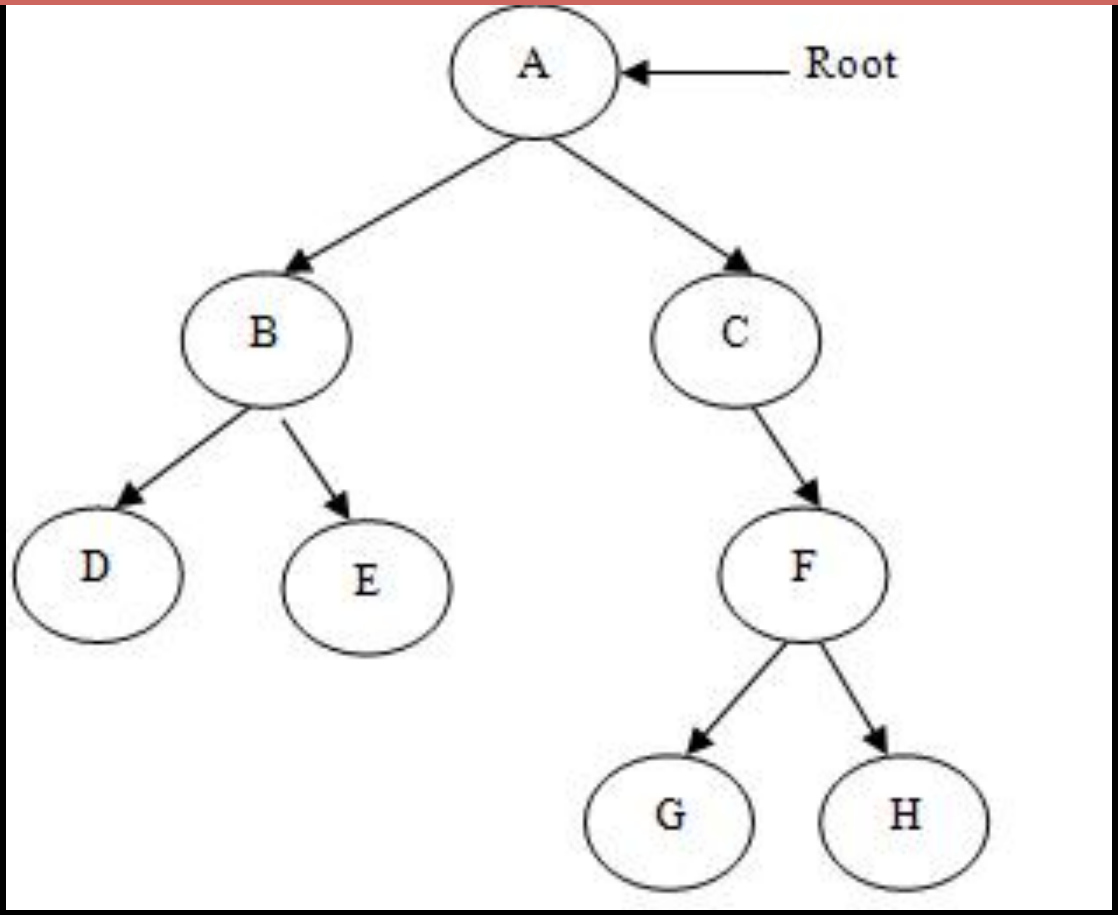
In-Order Traversal

D -> B -> E -> A -> C -> G -> F -> H



Post-Order Traversal

D -> E -> B -> G -> H -> F -> C -> A

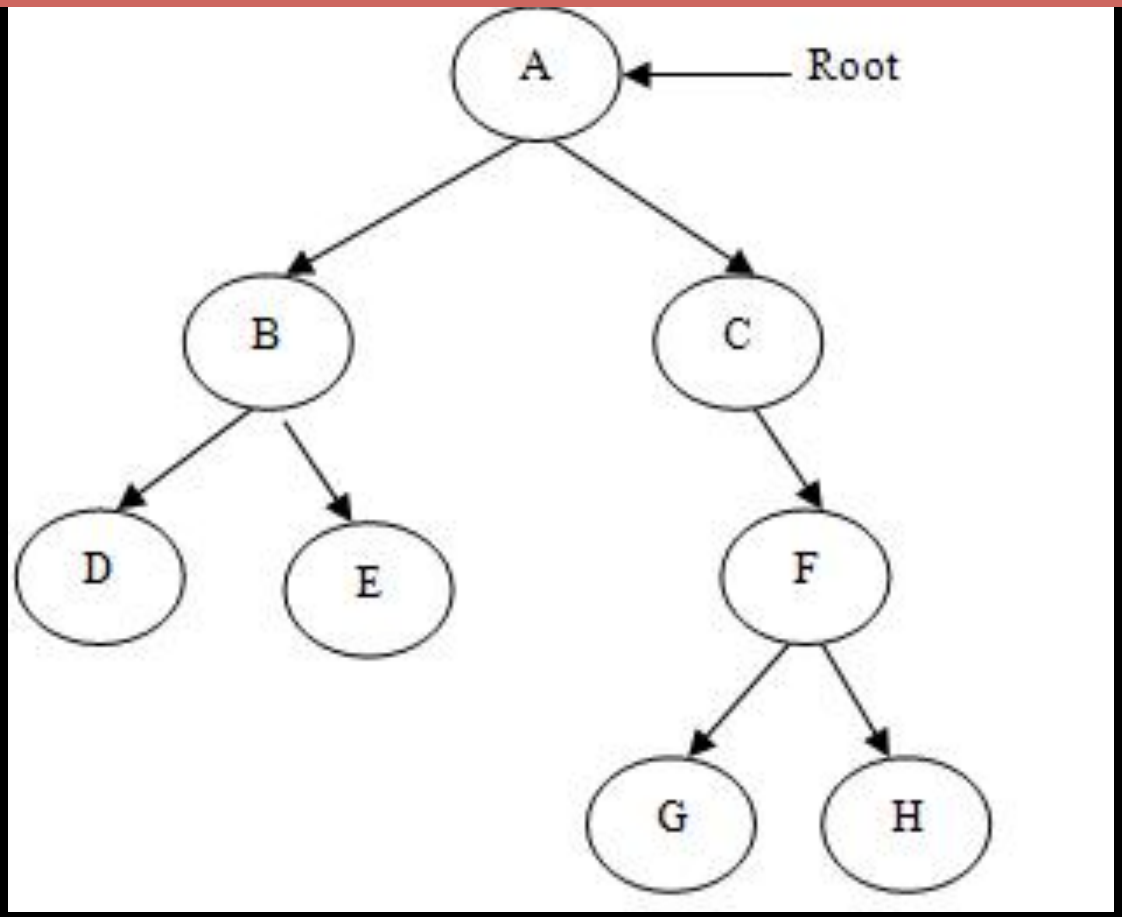


Breadth-first Traversal

- Cannot be done recursively
- Done iteratively with the help of a queue
  - You did this in lab 7

Breadth-first (queue lhs before rhs)

A -> B -> C -> D -> E -> F -> G -> H



Question

- Which of the traversals will output a sorted version of a binary search tree?
- When should you use recursion / iteration?
  - Most importantly: use what you're comfortable with
  - With recursion be careful of stack depth (O(log(n)) memory is okay, O(n) is probably not)
- CHALLENGE:
  - Implement the three depth-first traversals iteratively

LAB 7 RECURSIVE SOLUTION

Lab 7 Recursive ~BST

```
template <class T>
void bst_destruct(BinaryNode<T> *node) {
  if (node != NULL) {
    bst_destruct(node->get_rhs());
    bst_destruct(node->get_lhs());
    delete node;
  }
}

template <class T>
BST<T>::~BST() {
  bst_destruct(root);
}
```

What traversal order is this?

How much memory is required?

Lab 7 Recursive insert

```
template <class T>
bool BST<T>::insert(T item) {
  try {
    root = bst_insert(root, item);
    return true;
  }
  catch (int e) {
    return  false;
  }
}
```

Lab 7 Recursive insert

```
template <class T>
BinaryNode<T> *bst_insert(BinaryNode<T> *node, T item) {
  if (node == NULL)
    return new BinaryNode<T>(item, NULL, NULL);
  else if (item == node->get_data())
    throw 1;
  else if (item < node->get_data())
    node->set_lhs(bst_insert(node->get_lhs(), item));
  else
    node->set_rhs(bst_insert(node->get_rhs(), item));
  return node;
}
```

How much memory is required?

Save space by using const references where appropriate

```
template <class T>
BinaryNode<T> *bst_insert(BinaryNode<T> *node, const T &item) {
  if (node == NULL)
    return new BinaryNode<T>(item, NULL, NULL);
  else if (item == node->get_data())
    throw 1;
  else if (item < node->get_data())
    node->set_lhs(bst_insert(node->get_lhs(), item));
  else
    node->set_rhs(bst_insert(node->get_rhs(), item));
  return node;
}
```

MERGE SORT

O(nlog(n)) Sort Algorithms

- Merge Sort
  - Divide the problem in half until you have 1 or 2 items and put them in order
  - Merge the two sorted halves
- T(n) = 2T(n/2) + O(n) === O(n*log(n))
  - (masters theorem)

Coding Hint

- What's wrong with the following?

```
int *tmp = new int[10];
tmp = some_function();
```

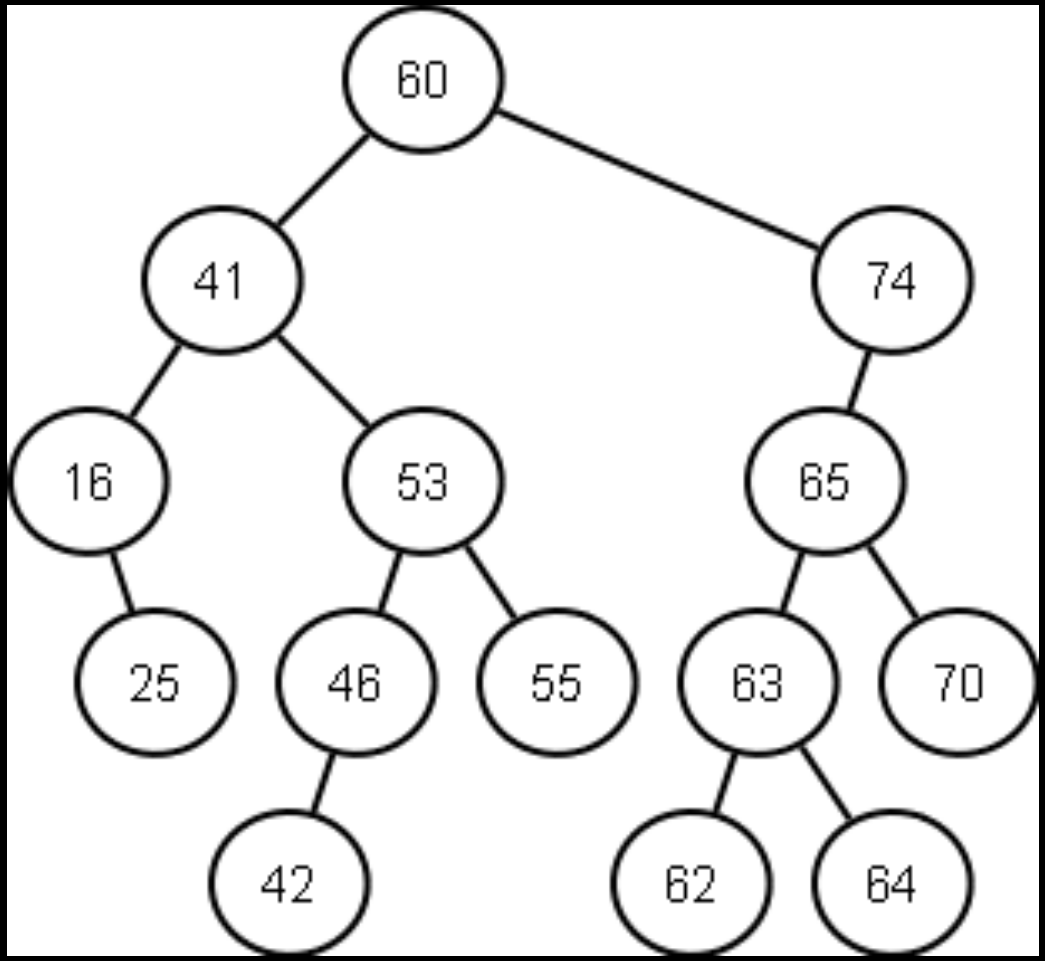
results in a memory leak

BST REMOVAL

Recall

- A binary search tree is a tree with the property that the value of all descendants of a node's left subtree are smaller, and the value of all descendants of a node's right subtree are larger

BST Example



BST Remove

- If the node has no children simply remove it
- If the node has a single child, update its parent pointer to point to its child and remove the node

Removing a node with two children

- Replace the value of the node with the largest value in its left-subtree (right-most descendant on the left hand side)
  - Note you could use the smallest value in the right-subtree (but for consistency please don't)
- Then repeat the remove procedure to remove the node whose value was used in the replacement

Removing a node with two children